컨테이너 기반 Smart Air IoT-Cloud 응용 서비스

Document No. KOREN #7
Version 1.0
Date 2017-11-24
Author(s) 김성환
이왕광
염성웅
임재학
최영수
김경백

■ 문서의 연혁

버전	날짜	작성자	비고
0.1	2017. 09. 18	김성환,이왕광,염성웅	초안
0.5	2017. 11. 08	김성환,이왕광,염성웅,임재학	1차 수정
0.7	2017. 11. 17	김성환,이왕광,염성웅,임재학,최영수	2차 수정
0.9	2017. 11. 29	김성환,이왕광,염성웅,임재학,최영수,김경백	3차 수정

본 연구는 한국정보화진흥원(NIA)의 미래네트워크연구시험망(KOREN) 사업 지원과제의 연구결과로 수행되었음 (2016-기술-위20)

This research was one of KOREN projects supported by National Information Society Agency (2016-기술-위20)

Contents

#7. 컨테이너 기반 Smart Air IoT-Cloud 응용 서비스

그림 목차

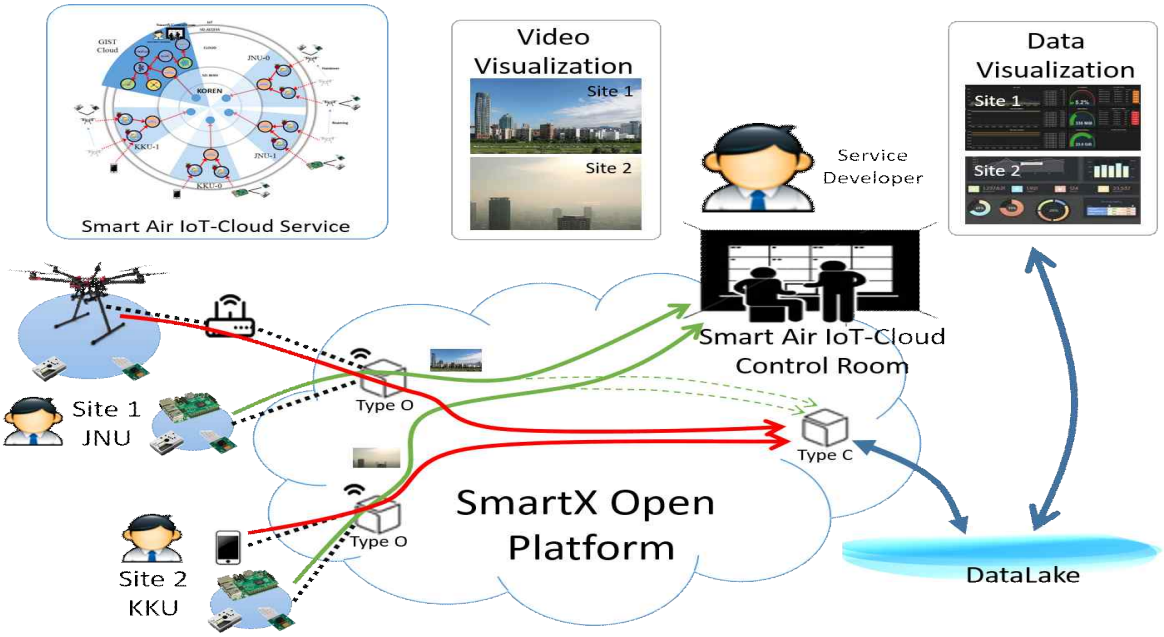
#7. 컨테이너 기반 Smart Air IoT-Cloud 응용 서비스

1. Smart Air IoT-Cloud 응용 서비스 개요

1.1 목적 및 개요

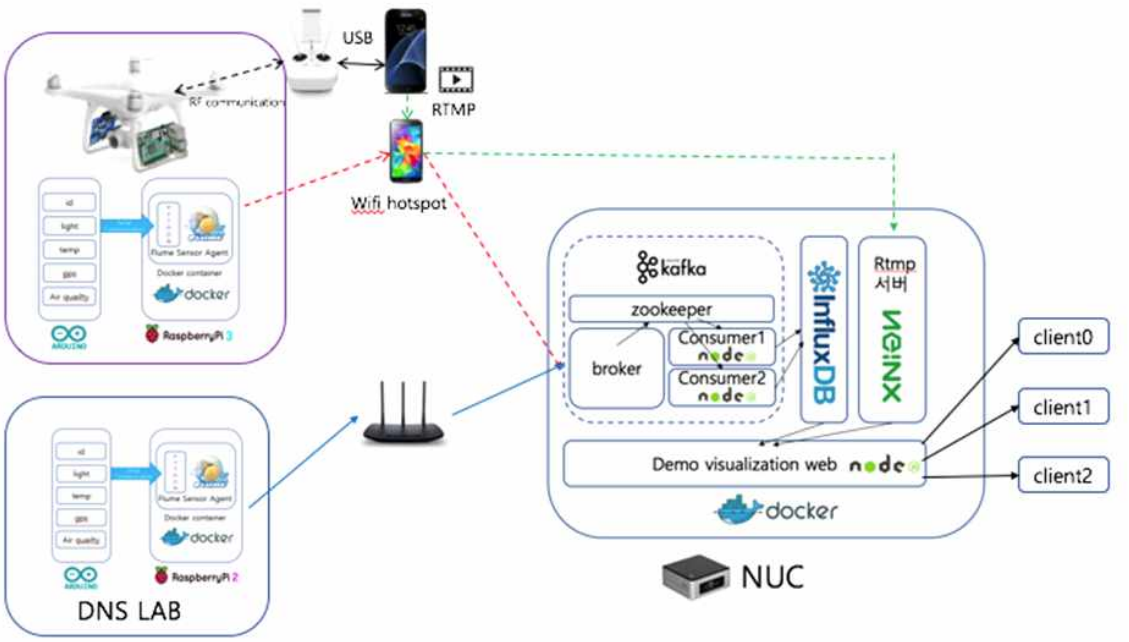
○ 본 문서에서는 KOREN SmartX Open Platform에서 제공하는 SDN/SDI OpenAPI를 활용하여 사용자가 원하는 서비스 운영에 필요한 클라우드/네트워크 자원을 확보하고 추가적인 요소기능을 투입하여 전체 서비스로 합성하는 일련의 응용서비스 지원 프로세스 실증사례인 Smart Air IoT-Cloud 응용서비스 구현방안을 기술한다

○ Smart Air IoT-Cloud 서비스는 고정식 센서 모듈 또는 드론 및 스마트폰을 활용한 이동식 센서 모듈을 통해 다수지역의 기상정보 및 미세먼지 정도를 실시간으로 측정하여 관련 데이터를 수집하고 관측내용을 분석하여 시각화하는 기능을 제공한다. 다수의 사이트에서 운용되는 드론 및 센서 모듈들에서 측정되는 실시간 센싱 데이터는 SmartX Open Platform을 통해 구성된 클라우드/네트워크 슬라이스를 이용하여 DataLake로 수집되고 저장된다. 각 서비스 사이트에서 운용되는 IoT 기기들은 Smart Air IoT-Cloud서비스를 위한 네트워크 슬라이스에 포함된 Type O SD-Access Box와 연결하여 데이터를 전송한다. DataLake는 다수의 IoT기기에서 제공되는 데이터를 검증하고 수집하기 위한 Smart IoT-Cloud Hub와 수집된 데이터의 저장 및 간략한 분석기능을 제공하는 SmartX Cloud로 구성된다. DataLake에 수집된 데이터를 Smart Air IoT-Cloud Control Room에서 효과적으로 분석하고 활용하기 위해, 현재 운용중인 서비스 모듈에서의 데이터 흐름 및 상태 현황을 손쉽게 파악하는 기능을 제공하는 서비스운용 시각화 기능 및 IoT데이터 현황을 다양한 측면에서 파악하기 위한 데이터분석 UI도구를 제공한다. 이러한 Smart Air IoT-Cloud 서비스를 KOREN SmartX Open Platform상에서 실증하기 위한 테스트베드를 구축하였고, 본 문서를 통해 테스트베드 구축에 관련된 자세한 사항을 설명한다.



<그림 1: Smart Air IoT-Cloud 응용서비스 실증 개념도>

1.2 데모 시스템 개요



<그림 2: Demo 시스템 구성도>

○ DNS LAB 내의 거치식 아두이노와 드론에 부착된 아두이노는 각각 빛, 온도, gps, 알코올, co-gas 값을 받는다.

○ 드론에 부착된 아두이노에서 라즈베리파이로 값을 보내고 라즈베리파이에 있는 파이썬에서 값을 파싱하여 Flume을 통해 서버로 전달한다.

○ DNS LAB에 있는 실내 거치식 아두이노에서 라즈베리파이로 값을 보내고 라즈베리파이에 있는 파이썬에서 값을 파싱하여 Flume을 통해 서버로 전달한다.

○ 서버에 있는 kafka내 broker에서 센싱된 값을 받아 저장하고 있다가 Node.js 기반의 consumer를 활용해 해당값을 주기적으로 가져가서 InfluxDB에 저장한다.

○ Node.js기반 demo visualization를 활용해 InfluxDB에 있는 값을 가져와 network flow를 보여준다.

○ 드론에서 받은 영상정보를 RTMP를 활용해 Nginx rtmp proxy 서버로 보내고 Demo visualization에서 해당 proxy서버의 비디오 스트림을 호출한다.

2. 필수 환경 설정

2.1. Drone 설정

1. 안드로이드 기반 휴대폰에 DJI4 GO어플을 설치한다.

2. DJI4 GO어플을 실행한다.

3. 드론 조종기와 휴대폰을 USB연결션으로 연결한 후 조종기의 파워를 켠다.

4. 드론 조종기와 휴대폰 연결이 확인 되면, 드론의 파워를 켠다.

5. Fly 버튼을 눌러 DJI4 GO 어플을 실행하여 드론과 조종기 간의 연결을 확인한다.

2.2. Docker 설정

○ 도커는 컨테이너 기반 가상화 도구로써, 계층화된 파일시스템을 사용해 가상화된 컨테이너의 변경사항을 모두 축적하고 관리한다. 또한 컨테이너의 특정 상태를 항상 보존해두고, 필요할 때 언제 어디서나 실행할 수 있도록 도와주는 어플리케이션이다.

○ 도커는 운영체제 위에 또 다른 운영체제 사용 시 발생하는 단점인 오버헤드를 극복하기 위한 가상화 어플리케이션이다. 또한 커널 영역을 공유하면서 격리된 환

경을 만들어주기 때문에 메모리 사용이 유연하다.

○ Docker의 설치 과정은 공식 홈페이지의 가이드에 따르며, 설치 이후 docker 유저 그룹에 사용자를 추가해야 root 권한 없이 docker 명령어를 사용할 수 있다.

다음 docker run을 통해 docker 설치를 실행한다. 이에 대한 내용은 다음과 같다.

```
wget -qO- https://get.docker.com/|sh   //docker 다운로드
service docker start                   //docker 실행
```

○ 도커에 유저정보를 추가해 유저가 sudo를 사용하지 않더라도 docker 명령어를 사용할 수 있게 한다. 이에 대한 내용은 다음과 같다.

```
adduser ${user_name} docker            //sudo없이 명령어 사용
```

3. Flume Kafka 기반 센서데이터 수집 서비스

3.1 아두이노와 라즈베리파이에서 센싱 및 수집

○ 라즈베리파이의 Flume에서 아두이노에서 수집되는 센서 값들을 데이터레이크로 보내려면 우선 라즈베리파이와 아두이노 간의 시리얼 통신이 필요하다.

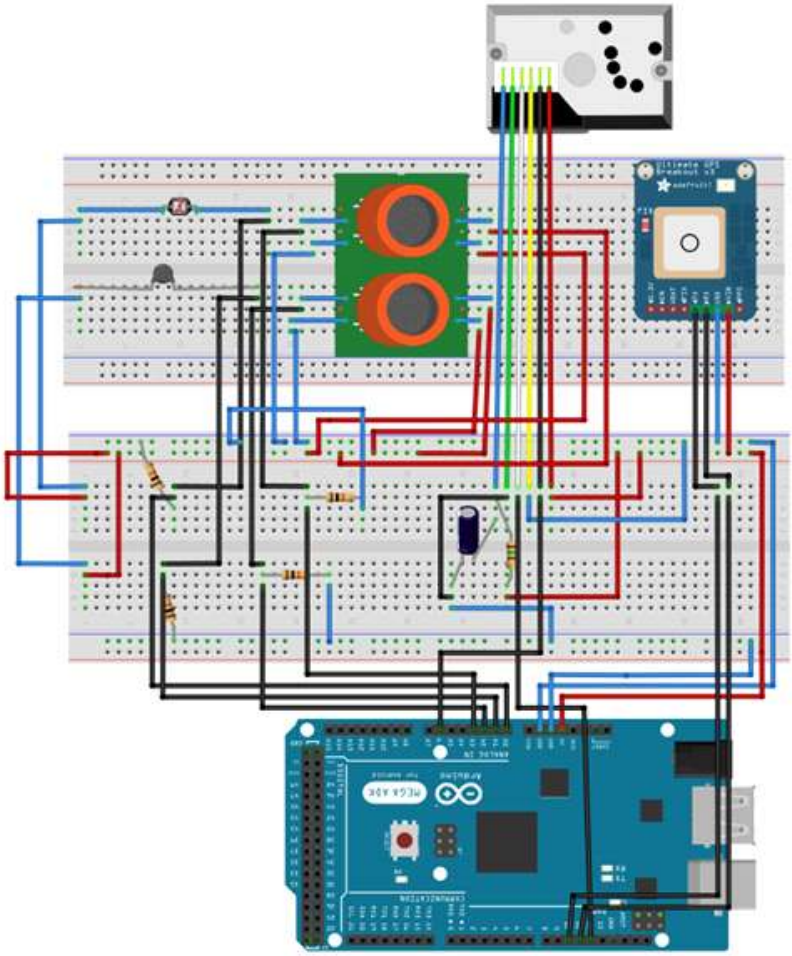
○ Arduino 센서

본 기술문서에서 사용한 센서 매트릭은 다음과 같다. GPS 센서에서 나오는 Latitude, Longitude 값과 알코올가스 센서에서의 alcohol_gas 값, 일산화탄소가스 센서에서의 carbon oxide gas 값, 온도센서에서 나오는 temperature 값, 조도센서에서 나오는 light 값, 먼지센서에서 나오는 dust값으로 총 6개 값을 사용한다.

no	name	type	range	error value	m/o	desc.
1	device_id	string			m	디바이스의 고유 ID
2	status	integer	[0,1,2]		m	0: Success 1: Sensing Error (일부에서 에러나 누락 발생시에도 1로 표시하고, 해당 센서 측정치에 error value 설정) 2: Comm. Error
3	time	datetime			m	저장된 시간
4	lat	float	-90.0~90.0		m	위도
5	lon	float	-180.0~180.0		m	경도
6	alcohol_gas	integer	0~1023	-1	o	알코올 가스량 (ppm)
7	co_gas	integer	0~1023	-1	o	일산화탄소량 (ppm)
8	temp	float	-40.0~125.0	-999	o	온도
9	light	integer	0~1023	-1	o	조도
10	dust	float	0.0~750.0		o	먼지(ug/m3)

<그림 3 : 센서 메트릭 구성>

○ 상기 센서들을 포함하는 Arduino 회로는 다음과 같이 구성된다.



<그림 4: Arduino 회로도>

○ 구성된 Arduino 회로의 각 센서들에서 값을 주기적으로 읽어오고 시리얼 통신으로 값을 보내기위해 다음과 같이 코드를 작성하였다. GPS 사용을 위하여 SoftwareSerial Library와 Adafruit_GPS 라이브러리를 이용하였다. 아두이노쪽 코드는 다음과 같다.

```
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

SoftwareSerial mySerial(11, 10);
Adafruit_GPS GPS(&mySerial);
```

```
int light;
float celsius;
int Alcohol_gas;
char pChrBuffer[50];
int Carbonmonoxide;
int measurePin = 6;
int ledPower = 12;
int samplingTime = 280;
int deltaTime = 40;
int sleepTime = 9680;
float voMeasured = 0;
float calcVoltage = 0;
float dustDensity = 0;

// Set GPSECHO to 'false' to turn off echoing the GPS data to Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences.
#define GPSECHO  false

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy

void setup()
{

  Serial.begin(115200);

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's
  GPS.begin(9600);

  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);    // 1 Hz update rate

  GPS.sendCommand(PGCMD_ANTENNA);
```

```
  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes
  // the loop code a heck of a lot easier!
  useInterrupt(true);

  delay(1000);
  // Ask for firmware version
  mySerial.println(PMTK_Q_RELEASE);
  pinMode(ledPower, OUTPUT);
}

float pad = 10000;
int p_count = 0;

float convertToCelsius(float RawADC) {
  long Resistance;
  float Temp; // Dual-Purpose variable to save space.
  Resistance = ((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) +
                          (0.0000000876741 * Temp * Temp * Temp));
  Temp = Temp - 273.15;
  return Temp;
}

// Interrupt is called once a millisecond, looks for any new GPS data,
// and stores it
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
  // writing direct to UDR0 is much much faster than Serial.print
  // but only one character can be written at a time.
#endif
```

```
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}

uint32_t timer = millis();
void loop()                     // run over and over again
{
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trying to print data
    // this also sets the newNMEAreceived() flag to false
    //Serial.println(GPS.lastNMEA());
```

```
    // this also sets the newNMEAreceived() flag to false
    if (!GPS.parse(GPS.lastNMEA()))
      // we can fail to parse a sentence
      // in which case we should just wait for another
      return;
  }
  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // approximately every second or so, print out the current stats
  if (millis() - timer > 1000) {
    timer = millis(); // reset the timer

    digitalWrite(ledPower, LOW); // power on the LED
    delayMicroseconds(samplingTime);

    voMeasured = analogRead(measurePin); // read the dust value

    delayMicroseconds(deltaTime);
    digitalWrite(ledPower, HIGH); // turn the LED off
    delayMicroseconds(sleepTime);

    float temp = 0;

    light = analogRead(0);
    temp = analogRead(1);
    Carbonmonoxide = analogRead(2);          // read analog input pin 0
    Alcohol_gas = analogRead(3);

    celsius = convertToCelsius(temp);
    dtostrf(celsius , 5, 2, pChrBuffer);

    digitalWrite(ledPower, LOW);
    delayMicroseconds(samplingTime);
    voMeasured = analogRead(measurePin); // read the dust value
    delayMicroseconds(deltaTime);
    digitalWrite(ledPower, HIGH); // turn the LED off
```

```
        delayMicroseconds(sleepTime);

        calcVoltage = voMeasured * (5.0 / 1024);

        dustDensity = (0.17 * calcVoltage - 0.1) * 1000;

        Serial.print(pChrBuffer);
        Serial.print(";");
        Serial.print(light);
        Serial.print(";");
        Serial.print(Carbonmonoxide);
        Serial.print(";");
        Serial.print(Alcohol_gas);
        Serial.print(";");
        Serial.print(dustDensity);
        Serial.print(";");
        Serial.print(GPS.latitudeDegrees, 5);
        Serial.print(";");
        Serial.print(GPS.longitudeDegrees, 5);
        Serial.println(";");
    }
}
```

○ 라즈베리파이3 사용 시 블루투스기능 때문에 시리얼 통신이 잘 안 되는 경우가 있다. 이 때 블루투스 기능을 사용하지 않으려면 /boot/config.txt에 다음과 같은 설정을 추가한다.

```
# Disable Bluetooth
dtoverlay=pi3-dsable-bt
```

○ 위 설정을 저장한 후, 블루투스 컨트롤러를 정지시켜야한다. 이에 대한 내용은 다음과 같다.

$ sudo systemctl disable hciuart

○ seri.py를 실행시키면 아두이노에서 시리얼통신으로 보낸 값을 ser.readline() 함수로 읽어오게 되고 open("<파일을 저장할 주소>" . 'a')함수와 write(<입력하고자 하는 data>) 함수를 통하여 파일이 있을시 추가하여 값을 저장하고, 해당파일이 없을 경우 파일을 새로 만들어서 값을 저장하게 된다. serial.Serial(

'/dev/ttyACM0' . <rate> timeout=<timeout 시간>) 에서 rate는 아두이노에서 설정한 baud rate값과 동일하게 해줘야 값을 제대로 받을 수 있다. 파이썬 코드 seri.py의 내용은 다음과 같다.

```
import serial
import time

nullstr = -1
id = '"RaspberryPi"'
light = nullstr
temp = nullstr
latitude = nullstr
longitude = nullstr
dust = nullstr
humidity = nullstr
luminance = nullstr
alcohol_gas = nullstr
co_gas = nullstr
ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)
if(ser.isOpen() == False):
    ser.open()
ser.readline()
time.sleep(2)
while 1:
        response = ser.readline()
#       print(response.split(';'))
        if len(response.split(';')) == 8 :
                temp = response.split(';')[0]
                light = response.split(';')[1]
                co_gas = response.split(';')[2]
                alcohol_gas = response.split(';')[3]
                dust = response.split(';')[4]
                latitude = response.split(';')[5]
                longitude = response.split(';')[6]
                f = open("/var/log/apache/flumeSpool/sensor.txt", 'a')
                data = '{ "ID" : %s , "light" : %s , "temp" : %s , "latitude"
: %s , "longitude" : %s , "dust" : %s , "alcohol_gas" : %s , "co_gas" : %s
```

```
}\n' %(id,light,temp,latitude,longitude,dust,alcohol_gas,co_gas)
                    f.write(data)
```

○ 파이썬 코드를 작성한 후 실행시켜준다. 이에 대한 내용은 다음과 같다.

```
$ sudo python seri.py
```

3.2 Flume

3.2.1 Docker Image 생성

○ Flume Docker Container를 생성하기 위해서는 Docker image가 필요하다. Docker image는 이미 만들어진 image를 다운 받거나 Dockerfile을 이용하여 image를 빌드하고 생성할 수 있다. Flume Docker image를 빌드하기 위한 Dockerfile의 내용은 아래와 같다.

```
FROM resin/rpi-raspbian:wheezy

#Update & Install wget, vim
RUN apt-get update
RUN apt-get -y install wget
RUN apt-get -y install vim
RUN apt-get -y install python

#Timezone
RUN cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime

#Install Oracle JAVA
RUN mkdir -p /opt
RUN curl -O -v -j -k -L -H "Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-arm32-vfp-hflt.tar.gz
RUN tar -xvzf jdk-8u131-linux-arm32-vfp-hflt.tar.gz -C /opt

#Configurate environmental variables
ENV JAVA_HOME /opt/jdk1.8.0_131
ENV PATH $PATH:/opt/jdk1.8.0_131/bin
```

```
RUN ln -s /opt/jdk1.8.0_131/bin/java /usr/bin/java

#Install Flume
RUN sudo wget --no-check-certificate http://www.apache.org/dist/flume/1.6.0/apache-flume-1.6.0-bin.tar.gz -O - ¦ tar -zxv
RUN sudo mv apache-flume-1.6.0-bin /flume

ADD plugins.d /flume/plugins.d
ADD flume-conf.properties /flume/conf/

#Csharp Programming
COPY seri.py /flume/
COPY autosensor.sh /flume/

#Working directory
WORKDIR /flume
```

○ 컨테이너 생성 후 Flume Configuration 파일, python 파일, 센서수집 파일이 필요하기 때문에 Dockerfile, flume-conf.properties, plugins.d seri.py, autosensor.sh를 가진 폴더에서 Dockerfile을 빌드한다. 이에 대한 내용은 다음과 같다.

$ sudo docker build --tag <이미지이름> .

```
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker build --
tag raspiflume .
Sending build context to Docker daemon  889.3kB
Step 1/19 : FROM resin/rpi-raspbian:wheezy
# Executing 1 build trigger...
Step 1/1 : RUN echo $'WARNING: Systemd is not available on this base image! \nSystemd was included
 in Debian wheezy as a technology preview. Therefore, we do not install systemd on wheezy images a
nd INITSYSTEM will not work.'
 ---> Using cache
 ---> 9f17093c921d
Step 2/19 : MAINTAINER Seungryong Kim <srkim@nm.gist.ac.kr>
 ---> Using cache
 ---> 83199c0f8531
Step 3/19 : RUN apt-get update
 ---> Using cache
 ---> 82fa70eed567
Step 4/19 : RUN apt-get -y install wget
 ---> Using cache
 ---> 582276e1768e
Step 5/19 : RUN apt-get -y install vim
 ---> Using cache
 ---> 8c30efe83684
Step 6/19 : RUN cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime
 ---> Using cache
 ---> 14b5164acd23
```

<그림 5: Dockerfile build 과정>

○ 도커 이미지 생성확인을 하는 방법은 다음과 같다.

$ sudo docker images

```
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
raspiflume          latest              8cebc0e000df        36 seconds ago      556MB
```

<그림 6: 생성한 flume docker image 확인 결과>

○ 이미지로 컨테이너 생성

컨테이너에서 시리얼통신 사용시 --privileged 옵션을 줘야지만 컨테이너에서 정상적으로 Serial 통신이 가능하다.

$ sudo docker run -it --privileged --net=host --name <컨테이너_이름> <사용할 이미지 이름>

○ 생성된 컨테이너 확인

$ sudo docker ps -a

```
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
6ec5f03f9f48        3b6ddfc939af        "/usr/bin/entry.sh..."   4 hours ago         Up 3 hours                              flume
```

<그림 7: 생성한 flume docker container 확인 결과>

3.2.2 Configuration Setting

○ flume에서 서버에 있는 kafka로 값을 보내기 위해서는 flume설정을 알맞게 바꿔줘야 한다. 이러한 flume의 설정파일에 대한 내용은 다음과 같다.

$sudo vi /flume/conf/flume-conf.properties

```
# Name the components on this agent
agent.sources = source1
agent.sinks = sink1
agent.channels = channel1

# The source1
agent.sources.source1.type = exec
agent.sources.source1.channels = channel1
agent.sources.source1.command = sh /flume/autosensor.sh
agent.sources.source1.fileHeader = true
```

```
# The channel
agent.channels.channel1.type = memory

# The sink1
agent.sinks.sink1.type = org.apache.flume.sink.kafka.KafkaSink
agent.sinks.sink1.topic = JNU01_rasp
agent.sinks.sink1.brokerList = gist:9092
agent.sinks.sink1.requiredAcks = 1
agent.sinks.sink1.batchSize = 1

# Bind the source and sink to the channel
agent.sources.source1.channels = channel1
agent.sinks.sink1.channel = channel1
```

○ 본 기술문서에서는 Flume의 source type을 exec으로 설정하여 쉘스크립트를 실행시켜서 읽어 들여 콘솔로 출력한 센서값들이 Flume의 source가 되도록 설정하였다. 이후 sink1.topic에 사용하고자 하는 <topic name>을 입력하고, sink1.brokerList에는 이용하고자 하는 Kafka 브로커의 <broker IP>:<broker port>를 입력한다.

○ 다음은 Flume에서 사용한 exec source file인 autosensor.sh의 내용이다.

```
#!/bin/sh
sensor="/var/log/apache/flumeSpool/sensor.txt"
temp="/var/log/apache/flumeSpool/sensor_Flume.txt"
while [ 1 ]
do
  if [ -e $sensor ]
    then
      mv /var/log/apache/flumeSpool/sensor.txt $temp
      cat $temp
      rm $temp
      sleep 2
  fi
done
```

○ 브로커에 대한 주소 및 포트 설정시 편의를 위해, /etc/hosts에 추가하고자 하는 host를 <host IP> <host name> 같은 형식으로 추가할 수 있다.

```
127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters

127.0.1.1       raspberrypi
168.131.148.45  drone
210.114.90.176  gist
```

<그림 8: /etc/hosts 예>

3.2.3 Container 실행

○ Docker container가 실행중일 때, attach 명령어를 통하여 container에 접속 가능하다. Ctrl + P + Q를 통하여 컨테이너를 끄지 않고 나올 수 있다.

$ sudo docker attach flume

```
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker attach flume
root@raspberrypi:/flume#
```

<그림 9: docker attach flume 결과>

○ Docker container가 꺼져있을 경우, start 명령어를 통하여 컨테이너를 실행시킨 다음 attach 명령어를 통하여 container에 접속한다.

$ sudo docker start flume
$ sudo docker attach flume

```
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker start flume
flume
root@raspberrypi:/home/pi/Services/Smart_Air_IoT_Cloud_Service/JNU/raspbian-flume# docker attach flume
root@raspberrypi:/flume#
```

<그림 10: docker start, attach flume 결과>

○ serial 통신포트를 설정한 후 flume agent실행한다. seri.py 파일을 이용하여 아두이노에서 보내는 센서값들을 라즈베리파이의 지정된 폴더안에 저장하고, agent를 실행시켜 센서값들을 kafka broker에게 전송한다. 이 때 kafka zookeeper와 broker는 실행 중이어야 한다. 이에 대한 내용은 다음과 같다.

$ sudo python seri.py &
$ sudo bin/flume-ng agent --conf conf --conf-file conf/flume-conf.properties
—name <flume agent name> -Dflume.root.logger=INFO,console

```
root@raspberrypi:/flume# bin/flume-ng agent --conf conf --conf-file conf/flume-conf.properties --name agent -Dflume.root.logger=INFO,console
Info: Including Hive libraries found via () for Hive access
+ exec /opt/jdk1.8.0_131/bin/java -Xmx20m -Dflume.root.logger=INFO,console -cp '/flume/conf:/flume/lib/*:/flume/plugins.d/snmp-source/lib/*:/flume/plugins.d/snmp-source/libext/*:/lib/*' -Djava.library.
path= org.apache.flume.node.Application --conf-file conf/flume-conf.properties --name agent
2017-09-20 07:01:44,416 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.node.PollingPropertiesFileConfigurationProvider.start(PollingPropertiesFileConfigurationProvider.java:61)] Configuration provi
der starting
2017-09-20 07:01:44,460 (conf-file-poller-0) [INFO - org.apache.flume.node.PollingPropertiesFileConfigurationProvider$FileWatcherRunnable.run(PollingPropertiesFileConfigurationProvider.java:133)] Reloa
ding configuration file:conf/flume-conf.properties
2017-09-20 07:01:44,528 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,529 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,531 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,533 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,534 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:931)] Added sinks: sink1 Agent: agent
2017-09-20 07:01:44,536 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,538 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration$AgentConfiguration.addProperty(FlumeConfiguration.java:1017)] Processing:sink1
2017-09-20 07:01:44,656 (conf-file-poller-0) [INFO - org.apache.flume.conf.FlumeConfiguration.validateConfiguration(FlumeConfiguration.java:141)] Post-validation flume configuration contains configurat
ion for agents: [agent]
2017-09-20 07:01:44,657 (conf-file-poller-0) [INFO - org.apache.flume.node.AbstractConfigurationProvider.loadChannels(AbstractConfigurationProvider.java:145)] Creating channels
2017-09-20 07:01:44,711 (conf-file-poller-0) [INFO - org.apache.flume.channel.DefaultChannelFactory.create(DefaultChannelFactory.java:42)] Creating instance of channel channel1 type memory
2017-09-20 07:01:44,754 (conf-file-poller-0) [INFO - org.apache.flume.node.AbstractConfigurationProvider.loadChannels(AbstractConfigurationProvider.java:200)] Created channel channel1
2017-09-20 07:01:44,761 (conf-file-poller-0) [INFO - org.apache.flume.source.DefaultSourceFactory.create(DefaultSourceFactory.java:41)] Creating instance of source source1, type exec
2017-09-20 07:01:44,826 (conf-file-poller-0) [INFO - org.apache.flume.sink.DefaultSinkFactory.create(DefaultSinkFactory.java:42)] Creating instance of sink: sink1, type: org.apache.flume.sink.kafka.Kaf
kaSink
```

<그림 11: flume agent 실행결과>

3.3. Kafka 설정

3.3.1 Docker image 생성

○ Kafka Docker Container를 생성하기 위해서는 Docker image가 필요하다. Docker image는 이미 만들어진 image를 다운 받거나 Dockerfile을 이용하여 image를 빌드하고 생성할 수 있다. Kafka Docker image를 빌드하기 위한 Dockerfile의 내용은 아래와 같다.

```
FROM ubuntu:14.04

#Update & Install wget
RUN sudo apt-get update
RUN sudo apt-get -y install wget

#Install Oracle JAVA
RUN sudo mkdir -p /opt
RUN sudo wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/8u45-b14/jdk-8u45-linux-x64.tar.gz" -O - | tar -zxv -C /opt

#Configurate environmental variables
ENV JAVA_HOME /opt/jdk1.8.0_45
ENV PATH $PATH:/opt/jdk1.8.0_45/bin
RUN ln -s /opt/jdk1.8.0_45/bin/java /usr/bin/java
```

```
#Install Kafka
RUN sudo wget --no-check-certificate http://apache.tt.co.kr/kafka/0.8.2.0/kafka_2.10-0.8.2.0.tgz -O - ¦ tar -zxv
RUN sudo mv kafka_2.10-0.8.2.0 /kafka

WORKDIR /kafka
```

○ 또는, 미리 생성된 kafka 이미지를 docker pull 명령어를 통해서 이미지를 다운받는다. 이 기술문서에서는 docker image를 다운받아 활용하는 내용을 다룬다.

$ sudo docker pull smartxenergy/kafka

○ 다운받은 docker image 확인

```
root@drone:/home/server# docker images | grep smartxenergy
smartxenergy/kafka   latest              7284d5844be7        7 months ago        593MB
```

<그림 12: 다운받은 docker image파일 확인 결과>

3.3.2. zookeeper container configuration & run

○ smartxenergy/kafka 이미지를 사용하여 zookeeper 컨테이너를 생성하고 확인한다. 이에 대한 내용은 다음과 같다.

$ sudo docker run -it --net=host --name <zookeeper name> <image name>

$ sudo docker ps

```
430082f48059   smartxenergy/kafka   "/bin/bash"   7 days ago   Up 7 days   zookeeper
```

<그림 13: zookeeper 컨테이너 실행 확인>

○ config/zookeeper.properties 파일을 통하여 zookeeper를 설정한다. 기본포트는 2181로 설정한다.

$ sudo vi config/zookeeper.properties

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since this is a non-production config
maxClientCnxns=0
~
```

<그림 14: zookeeper 설정 내용>

○ kafka broker/consumer, flume agent를 실행시키기 전 가장 먼저 zookeeper를 실행시켜야 한다. Zookeeper를 실행하는 방법은 다음과 같다.

$sudo bin/zookeeper-server-start.sh config/zookeeper.properties

```
root@drone:/kafka# bin/zookeeper-server-start.sh config/zookeeper.properties
[2017-09-20 06:18:59,974] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2017-09-20 06:18:59,976] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2017-09-20 06:18:59,976] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2017-09-20 06:18:59,976] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2017-09-20 06:18:59,976] WARN Either no config or no quorum defined in config, running  in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2017-09-20 06:18:59,993] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2017-09-20 06:18:59,993] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2017-09-20 06:19:00,001] INFO Server environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT (org.apache.zookeeper.server.ZooKeeperServer)
[2017-09-20 06:19:00,001] INFO Server environment:host.name=drone (org.apache.zookeeper.server.ZooKeeperServer)
[2017-09-20 06:19:00,001] INFO Server environment:java.version=1.8.0_45 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-09-20 06:19:00,001] INFO Server environment:java.vendor=Oracle Corporation (org.apache.zookeeper.server.ZooKeeperServer)
[2017-09-20 06:19:00,001] INFO Server environment:java.home=/opt/jdk1.8.0_45/jre (org.apache.zookeeper.server.ZooKeeperServer)
```

<그림 15: zookeeper 실행 결과>

3.3.3 broker container configuration & run

○ Zookeeper를 실행시킨 후에는 메시지 처리를 위한 Kafka Broker를 실행한다. 이를 위해 우선 broker docker container를 생성하고 확인한다. 이에 대한 내용은 다음과 같다.

$ sudo docker run -it --net=host —name <broker name> <image name>
$ sudo docker ps

```
root@drone:/home/server# docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
2a630fa41520        smartxenergy/kafka     "/bin/bash"         About a minute ago  Up About a minute                       broker
```

<그림 16: 생성한 broker 컨테이너 확인>

○ config/server.properties 파일을 통하여 kafka broker의 세부 설정을 수행할 수 있다. broker의 기본 port는 9092, loclhost이다. localhost가 아닌 외부에서 해당 broker 사용이 가능하게 하려면 advertised.host.name 부분에 <외부 IP>:<port>를 설정하고, 이 항목을 활성화 한다. 이에 대한 내용은 다음과 같다.

$ sudo vi config/server.properties

```
############################# Socket Server Settings #############################

# The port the socket server listens on
port=9092

# Hostname the broker will bind to. If not set, the server will bind to all interfaces
#host.name=localhost

# Hostname the broker will advertise to producers and consumers. If not set, it uses the
# value for "host.name" if configured.  Otherwise, it will use the value returned from
# java.net.InetAddress.getCanonicalHostName().
advertised.host.name=168.131.148.45

# The port to publish to ZooKeeper for clients to use. If this is not set,
# it will publish the same port that the broker binds to.
advertised.port=9092

# The number of threads handling network requests
num.network.threads=3

# The number of threads doing disk I/O
num.io.threads=8
```

<그림 17: broker config-1>

○ Kafka broker를 관리하는 zookeeper를 설정하기 위해서, configuration file의 zookeeper.connect=<zookeeper IP>:<zookeeper port>에 실행중인 zookeeper docker container의 IP와 port를 기입하고 해당 항목을 활성화 한다.

```
############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
```

<그림 18: broker config-2>

○ config/server.properties 파일에 설정된 대로 kafka broker를 실행한다. 이때, kafka broker 실행 전에는 zookeeper가 반드시 실행 중인지 확인해야 한다. Kafka broker를 실행시키는 방법은 다음과 같다.

$ sudo bin/kafka-server-start.sh config/server.properties

```
root@drone:/kafka# bin/kafka-server-start.sh config/server.properties
[2017-09-20 06:31:49,674] INFO Verifying properties (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,706] INFO Property broker.id is overridden to 0 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,706] INFO Property log.cleaner.enable is overridden to false (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,706] INFO Property log.dirs is overridden to /tmp/kafka-logs (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,707] INFO Property log.retention.check.interval.ms is overridden to 300000 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,707] INFO Property log.retention.hours is overridden to 168 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,707] INFO Property log.segment.bytes is overridden to 1073741824 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,707] INFO Property num.io.threads is overridden to 8 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,708] INFO Property num.network.threads is overridden to 3 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,708] INFO Property num.partitions is overridden to 1 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,708] INFO Property num.recovery.threads.per.data.dir is overridden to 1 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,708] INFO Property port is overridden to 9092 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,709] INFO Property socket.receive.buffer.bytes is overridden to 102400 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,709] INFO Property socket.request.max.bytes is overridden to 104857600 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,709] INFO Property socket.send.buffer.bytes is overridden to 102400 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,709] INFO Property zookeeper.connect is overridden to localhost:2181 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,710] INFO Property zookeeper.connection.timeout.ms is overridden to 6000 (kafka.utils.VerifiableProperties)
[2017-09-20 06:31:49,735] INFO [Kafka Server 0], starting (kafka.server.KafkaServer)
[2017-09-20 06:31:49,737] INFO [Kafka Server 0], Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
```

<그림 19: broker 실행 결과>

3.3.4 Topic 생성

○ topic은 zookeeper가 관리하며 zookeeper를 통하여 토픽을 생성 및 삭제가 가능하다. --partitions 옵션으로 topic 파티션 개수를 지정할 수 있다. flume agent에서 사용하기 위해 설정된 topic을 zookeeper에서 생성한다. 이를 위해 아래의 코드를 이용한다.

$ sudo bin/kafka-topics.sh --create --zookeeper <zookeeper hostname>:<zookeeper port> --replication-factor 1 --partitions <partition 개수> —topic <topic name>

○ zookeeper가 가지고 있는 topic을 확인하려면 --list 명령어를 통해 다음과 같이 확인이 가능하다.

$ sudo bin/kafka-topics.sh --list --zookeeper <zookeeper hostname>:<zookeeper port>

```
root@drone:/kafka# bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

<그림 20: zookeeper가 가지고 있는 topic list확인>

3.4 InfluxDB

○ InfluxDB는 TSDB(Terrorist Screening Database)의 일종으로 처리에 복잡한 논리, 분석 또는 비즈니스 규칙 및 데이터에 대한 처리에 유용하다. 전통적인 RDB(Relational Database)에서는 시간 범위 및 롤업 및 임의의 시간대 변환으로 가득차 있는 기록 데이터 쿼리가 어렵다.

○ TSDB 중 InfluxDB는 설치가 간편하며 검색할 때 SQL 구문을 활용할 수 있고 모니터링 도구인 Graphite 프로토콜을 지원한다.

3.4.1 Docker Image 생성

○ InfluxDB docker container는 InfluxDB official image를 이용해 생성한다. 이를 위해 다음의 명령을 이용해 InfluxDB official image를 받는다.

```
$ sudo docker pull influxdb
```

3.4.2 Container 생성

○ InfuxDB docker container를 생성할 때, 세부 설정사항들은 influxdb.conf에 기술한다. 본 서비스에서는 다음과 같은 설정파일을 사용한다.

```
reporting-disabled = false
bind-address = ":8088"

[meta]
  dir = "/var/lib/influxdb/meta"
  retention-autocreate = true
  logging-enabled = true

[data]
  dir = "/var/lib/influxdb/data"
  wal-dir = "/var/lib/influxdb/wal"
  query-log-enabled = true
  cache-max-memory-size = 1073741824
  cache-snapshot-memory-size = 26214400
  cache-snapshot-write-cold-duration = "10m0s"
```

```
  compact-full-write-cold-duration = "4h0m0s"
  max-series-per-database = 1000000
  max-values-per-tag = 100000
  trace-logging-enabled = false

[coordinator]
  write-timeout = "10s"
  max-concurrent-queries = 0
  query-timeout = "0s"
  log-queries-after = "0s"
  max-select-point = 0
  max-select-series = 0
  max-select-buckets = 0

[retention]
  enabled = true
  check-interval = "30m0s"

[shard-precreation]
  enabled = true
  check-interval = "10m0s"
  advance-period = "30m0s"

[admin]
  enabled = true
  bind-address = ":8083"
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"

[monitor]
  store-enabled = true
  store-database = "_internal"
  store-interval = "10s"

[subscriber]
  enabled = true
  http-timeout = "30s"
```

```
  insecure-skip-verify = false
  ca-certs = ""
  write-concurrency = 40
  write-buffer-size = 1000

[http]
  enabled = true
  bind-address = ":8086"
  auth-enabled = false
  log-enabled = true
  write-tracing = false
  pprof-enabled = true
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"
  https-private-key = ""
  max-row-limit = 0
  max-connection-limit = 0
  shared-secret = ""
  realm = "InfluxDB"
  unix-socket-enabled = false
  bind-socket = "/var/run/influxdb.sock"

[[graphite]]
  enabled = false
  bind-address = ":2003"
  database = "graphite"
  retention-policy = ""
  protocol = "tcp"
  batch-size = 5000
  batch-pending = 10
  batch-timeout = "1s"
  consistency-level = "one"
  separator = "."
  udp-read-buffer = 0

[[collectd]]
  enabled = false
```

```
  bind-address = ":25826"
  database = "collectd"
  retention-policy = ""
  batch-size = 5000
  batch-pending = 10
  batch-timeout = "10s"
  read-buffer = 0
  typesdb = "/usr/share/collectd/types.db"
  security-level = "none"
  auth-file = "/etc/collectd/auth_file"

[[opentsdb]]
  enabled = false
  bind-address = ":4242"
  database = "opentsdb"
  retention-policy = ""
  consistency-level = "one"
  tls-enabled = false
  certificate = "/etc/ssl/influxdb.pem"
  batch-size = 1000
  batch-pending = 5
  batch-timeout = "1s"
  log-point-errors = true

[[udp]]
  enabled = false
  bind-address = ":8089"
  database = "udp"
  retention-policy = ""
  batch-size = 5000
  batch-pending = 10
  read-buffer = 0
  batch-timeout = "1s"
  precision = ""

[continuous_queries]
  log-enabled = true
```

```
enabled = true
run-interval = “1s“
```

○ InfuxDB container를 생성하기위해 influxdb.conf파일이 있는 위치로 이동하여 다음의 명령을 수행한다.

$docker run -d --name influx_test influxdb

○ 여기서 쓰인 docker run의 매개변수의 의미는 다음과 같다. -d는 분리 모드에서 컨테이너를 실행시키는 것을 의미한다.

3.5 Consumer 설정

○ 해당 서비스에서 Flume을 통해 수집된 정보는 Kafka Broker에게 전달되고, Kafka Consumer를 통해 InfluxDB에 저장된다. 이를 위해 Kafka Consumer는 Node.js를 이용해 kafka에서 받은 값을 InfluxDB에 저장한다.

○ Node.js는 Chrome V8 JavaScript엔진으로 빌드된 JavaScript 런타임이다. Node.js는 이벤트 기반, 논 블로킹 I/O모델을 사용해 가볍고 효율적이다. 또한 Kafka관련 라이브러리도 포함하고 있어서 Node.js에서 Kafka를 구현하고 컨트롤하기 쉽다.

3.5.1 Docker Image 생성

○ Consumer Docker Image는 Node.js에 kafka consumer에서 받은 데이터를 influxDB에 저장하는 javascript를 올려서 실행하는 설정을 담고 있다.

○ 이를 위해, 우선 ubuntu-kafkatodb에 저장된 DockerFile의 내용을 설정에 맞게 수정한다. Dockerfile은 Node.js를 사용하기 위해 설정한 Docker image를 생성하기 위한 스크립트이다. Dockerfile안에 4가지를 설정해 주어야한다. server_ip는 사용하는 서버의 ip를 입력한다. influxdb_user_id는 influxdb의 새로 생성할 유저 아이디를 입력한다. influxdb_user_password는 influxdb의 새로 생성할 유저 패스워드를 입력한다. databases_name은 생성할 influxdb의 데이터베이스이름을 입력한다. 이에 대한 내용은 다음과 같다.

$ vim ~/Services/Smart_Air_IoT_Cloud_Service/JNU/ubuntu-kafkatodb/Dockerfile

```
FROM node:7
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
```

```
RUN curl -G -XPOST http://${server_ip}:8086/query --data-urlencode \“u=${influxdb_user_id}\“ --data-urlencode \“p=${influxdb_user_password}\“ --data-urlencode “q=CREATE USER ${influxdb_user_id} WITH PASSWORD '${influxdb_user_password}' WITH ALL PRIVILEGES“
RUN curl -G -XPOST http://${server_ip}:8086/query --data-urlencode \“u=${influxdb_user_id}\“ --data-urlencode \“p=${influxdb_user_password}\“ --data-urlencode “q=CREATE DATABASE ${database_name}“

COPY package.json /usr/src/app/
RUN npm install

COPY . /usr/src/app

CMD [“npm“,“start“]

EXPOSE 3000
```

○ 그 후에 ubuntu-kakfkatodb에 있는 kafka_to_db.js파일 내용을 Dockerfile에서 설정한 것과 맞게 수정한다. Docker image가 Container로 실행되면 kafka_to_db.js가 실행된다.

○ server_ip, influxdb_user_id, influxdb_user_password, databases_name 이 4가지 변수들은 앞서 Dockerfile에서 설정했던 변수들과 똑같이 설정하면 된다. 이에 대한 내용은 다음과 같다.

$ vim ~/Services/Smart_Air_IoT_Cloud_Service/JNU/ubuntu-kafkatodb/kafka_to_db.js

```
const influx = require('influx')
var kafka = require('kafka-node');

var topic_name = process.env.TOPIC_NAME
console.log('start');
// InfluxDB
var DB = new influx.InfluxDB({
    // single-host configuration
    host: '${server_ip}',
    port: 8086, // optional, default 8086
    protocol: 'http', // optional, default 'http'
    username: '${influx_user_id}',
```

```
    password: '${influx_user_password}',
    database: '${database_name}'
});

var resourceKafka = new kafka.Client('${server_ip}:2181');
var resourceOffset = new kafka.Offset(resourceKafka);

resourceOffset.fetch([{
        topic: topic_name,
        partition: 0,
        time: -1,
        maxNum: 1
    }
], function(err, data) {
    console.log(data);
    var resourceConsumer = new kafka.Consumer(resourceKafka, [{
            topic: topic_name,
            partition: 0,
            offset: data[topic_name][0]
        }
    ], {
        autoCommit: false,
        fromOffset: true
    });
    resourceConsumer.on('message', function(message) {
        var messageJSON = JSON.parse(message.value);
        DB.writePoints([{
            measurement: topic_name,
            tags: {
            },
            fields: {
               // 예시
                id : messageJSON.ID,
                light: messageJSON.light,
                temp: messageJSON.temp,
                latitude: messageJSON.latitude,
                longitude: messageJSON.longitude,
```

```
                    dust: messageJSON.dust,
                    alcohol_gas: messageJSON.alcohol_gas,
                    co_gas: messageJSON.co_gas
                },
            }])
            console.log(messageJSON);
        });
});
```

○ Dockerfile과 kafka_to_db.js설정을 완료한 후 도커 이미지를 아래와 같은 방법으로 빌드한다.

$ cd ~/Services/Smart_Air_IoT_Cloud_Service/JNU/ubuntu-kafkatodb/

$ docker build -t kafkatodb .

3.5.2 Container 실행

○ 빌드된 Docker image를 활용해 Docker Container를 실행 할 수 있다. 실행하는 방법은 다음과 같다.

$ docker run -it --net=host -e TOPIC_NAME=${topic_name} --name kafkatodb kafkatodb

여기서 쓰인 docker run의 매개변수의 의미는 다음과 같다. -it는 컨테이너와 상호적으로 주고받으면서, 터미널과 비슷한 환경을 조성해주는 의미이다. --net=host는 네트워크 연결을 host와 동일하게 하는 것이다. -e TOPIC_NAME=${topic_name}은 node를 실행시킬 때 TOPIC_NAME이라는 환경변수를 추가하여 실행시키는 것이다. ${topic_name}에는 설정한 topic이름을 넣으면 된다. --name kafkatodb은 컨테이너의 이름을 kafkatodb라고 설정하는 것이다. 어떤 이름을 설정하든 상관없다.

3.6 검증 방법 및 결과 확인

○ 해당서비스의 동작의 검증은 influxDB에 저장된 값들을 요청하여 해당값들이 잘 저장되어 있는지를 확인한다.

$ curl -G 'http://${server_ip}:8086/query?pretty=true' --data-urlencode "db=${database_name}" --data-urlencode "q=SELECT * FROM ${topic_name}"

여기서 쓰인 변수 3가지의 설명은 다음과 같다. server_ip와 databases_name은 Dockerfile에서 설정한 값을 넣어준다. topic_name은 docker run할 때 환경변수로 넣었던 인자를 넣어준다.


```
{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "JNU_rasp",
                    "columns": [
                        "time",
                        "alcohol_gas",
                        "co_gas",
                        "dust",
                        "id",
                        "latitude",
                        "light",
                        "longitude",
                        "temp"
                    ],
                    "values": [
                        [
                            "2017-09-18T08:52:56.794724369Z",
                            9,
                            117,
                            -0.05,
                            "RaspberryPi",
                            35.17959,
                            554,
                            126.90822,
                            26.52
                        ],
                        [
                            "2017-09-18T08:52:56.990976111Z",
                            9,
                            118,
                            -0.05,
                            "RaspberryPi",
                            35.17959,
                            541,
                            126.90822,
                            26.52
                        ],
                        [
                            "2017-09-18T08:52:56.991077571Z",
                            9,
                            118,
                            -0.05,
                            "RaspberryPi",
                            35.17959,
                            532,
                            126.90822,
                            26.61
                        ],
```


<그림 21: influxdb값 확인>

4. RTMP기반 실시간 비디오 스트리밍 서비스

4.1 스트리밍 서버 설정

○ 스트리밍 서버는 Nginx을 이용해서 구현한다. Nginx에서는 rtmp프록시 서버를 간단하게 구현할 수 있다. 또한 http도 같이 사용할 수 있기 때문에 rtmp 스트리밍 정보들을 http를 통해서도 확인이 가능하다.

4.1.1 Docker Image 생성

○ 다음 path로 이동한 후 Dockerfile을 이용하여서 Docker Image를 생성한다. 세부 설정을 수정할 경우 Dockefile과 nginx.conf파일을 수정하면 된다. nginx.conf파일은 nginx의 설정내용을 담고 있다. 아래와 같은 방법을 이용한다.

```
$ cd ~/Services/Smart_Air_IoT_Cloud_Service/JNU/nginx_rtmp_do
cker/
$ docker build -t nginx_rtmp .
```

○ Dockerfile의 세부내용은 아래와 같다.

```
FROM ubuntu:trusty

ENV DEBIAN_FRONTEND noninteractive
ENV PATH $PATH:/usr/local/nginx/sbin

EXPOSE 1935
EXPOSE 80

# create directories
RUN mkdir /src /config /logs /data /static

# update and upgrade packages
RUN apt-get update && \
  apt-get upgrade -y && \
  apt-get clean && \
  apt-get install -y --no-install-recommends build-essential \
  wget software-properties-common && \
# ffmpeg
```

```
  add-apt-repository ppa:mc3man/trusty-media && \
  apt-get update && \
  apt-get install -y --no-install-recommends ffmpeg && \
# unzip
  apt-get install -y --no-install-recommends unzip && \
# nginx dependencies
  apt-get install -y --no-install-recommends libpcre3-dev \
  zlib1g-dev libssl-dev wget && \
  rm -rf /var/lib/apt/lists/*

# get nginx source
WORKDIR /src
RUN wget http://nginx.org/download/nginx-1.13.1.tar.gz && \
  tar zxf nginx-1.13.1.tar.gz && \
  rm nginx-1.13.1.tar.gz && \
# get nginx-rtmp module
  wget https://github.com/arut/nginx-rtmp-module/archive/master.zip && \
  unzip master.zip && \
  rm master.zip

# compile nginx
WORKDIR /src/nginx-1.13.1
RUN ./configure --add-module=/src/nginx-rtmp-module-master \
  --conf-path=/config/nginx.conf \
  --error-log-path=/logs/error.log \
  --http-log-path=/logs/access.log && \
  make && \
  make install

ADD nginx.conf /config/nginx.conf
ADD static /static

WORKDIR /
CMD "nginx"
```

○ nginx.conf 세부내용은 아래와 같다.

```
daemon   off;

events {
    worker_connections 1024;
}

error_log stderr;

rtmp {
    server {
        listen 1935;
        chunk_size 4000;

        application encoder {
            live on;

            exec ffmpeg -i rtmp://localhost:1935/encoder/$name
              -c:a libfdk_aac -b:a 128k -c:v libx264 -b:v 400k -f flv -g 30 -r
30      -s      426x240      -preset      superfast      -profile:v      baseline
rtmp://localhost:1935/hls/$name_240p528kbs
              -c:a libfdk_aac -b:a 128k -c:v libx264 -b:v 2500k -f flv -g 30
-r     30     -s     1280x720     -preset     superfast     -profile:v     baseline
rtmp://localhost:1935/hls/$name_720p2628kbs;
        }

        application hls {
            live on;
            hls on;
            hls_fragment_naming system;
            hls_fragment 5s;
            hls_path /data/hls;
            hls_nested on;

            hls_variant                                          _720p2628kbs
BANDWIDTH=2628000,RESOLUTION=1280x720;
            hls_variant                                          _480p1128kbs
```

```
BANDWIDTH=1128000,RESOLUTION=854x480;
            hls_variant _360p878kbs BANDWIDTH=878000,RESOLUTION=640x360;
            hls_variant _240p528kbs BANDWIDTH=528000,RESOLUTION=426x240;
            hls_variant _240p264kbs BANDWIDTH=264000,RESOLUTION=426x240;
        }
    }
}

http {
    server {
        listen 80;

        location /hls {
            types {
                application/vnd.apple.mpegurl m3u8;
                video/mp2t ts;
            }
            root /data;
            add_header Cache-Control no-cache;
            add_header Access-Control-Allow-Origin * always;
        }

        location /stat {
            rtmp_stat all;
            rtmp_stat_stylesheet static/stat.xsl;
        }

        location /static {
            alias /static;
        }

        location /crossdomain.xml {
            default_type text/xml;
            return 200 '<?xml version="1.0"?>
                <!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
                <cross-domain-policy>
```

```
                    <site-control permitted-cross-domain-policies="all"/>
                    <allow-access-from domain="*" secure="false"/>
                    <allow-http-request-headers-from domain="*" headers="*"
secure="false"/>
                </cross-domain-policy>';
            expires 24h;
        }
    }
}
```

4.1.2 Docker Container 실행

○ 다음 docker run을 통해 nginx Docker image를 실행한다. 이에 대한 내용은 다음과 같다.

```
$ docker run -it -p 1935:1935 -p 1936:80 —name nginx_rtmp nginx_rtmp
```

여기서 쓰인 docker run의 매개변수의 의미는 다음과 같다. -it는 컨테이너와 상호적으로 주고받으면서, 터미널과 비슷한 환경을 조성해주는 의미이다. -p 1936:80는 컨테이너 내부에서 80포트사용을 외부에서는 1936포트로 연결한다는 의미이다. 1935포트는 rtmp포트이고 1936포트는 http포트이다.

4.1.3 검증 방법 및 결과 확인

○ 컨테이너가 실행되면 nginx의 http서버에 접속하여 nginx가 정상적으로 작동하고 있는지 확인한다. server_ip는 nginx Container를 실행시킨 서버의 아이피이다.

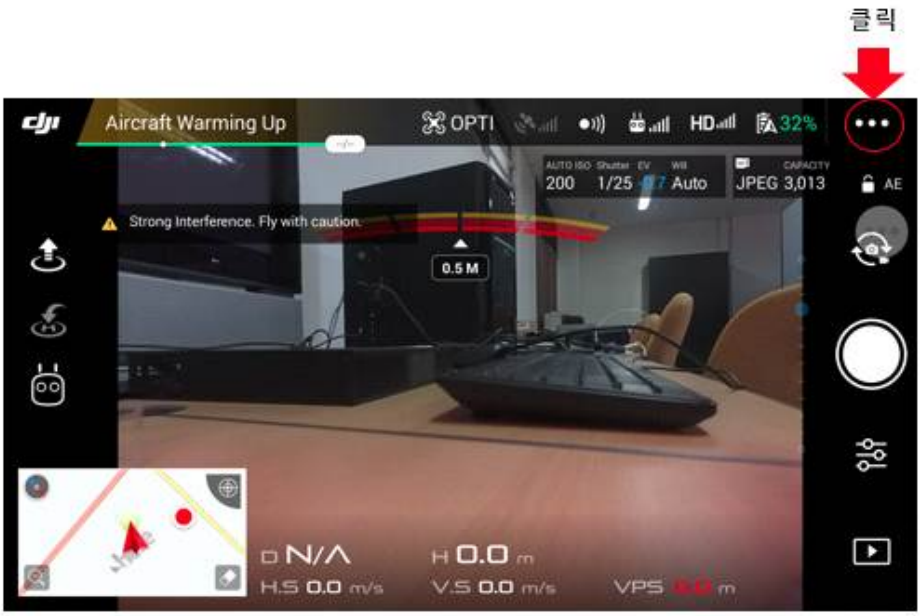
http://${server_ip}:1936/

<그림 22: nginx 실행 테스트 화면>

4.2 드론 스트리밍 설정

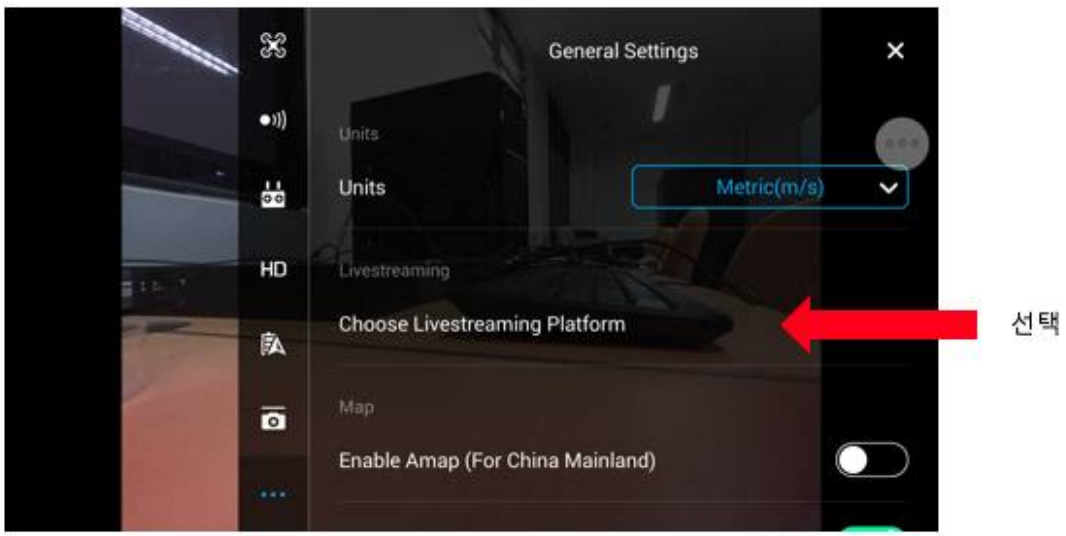
○ 드론 스트리밍은 드론을 조종하는 안드로이드 기기에서 다음과 같은 순서로 설정을 해주면 된다.

○ 오른쪽 위에 있는 메뉴버튼 클릭



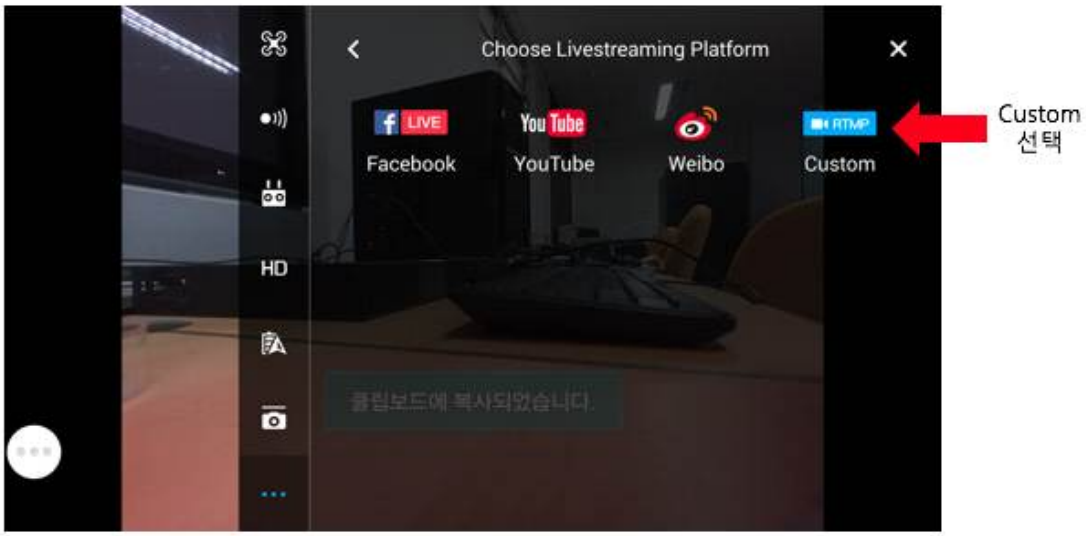
<그림 23: 드론 스트리밍 설정 방법-1>

○ Choose Livestreaming Platform 선택

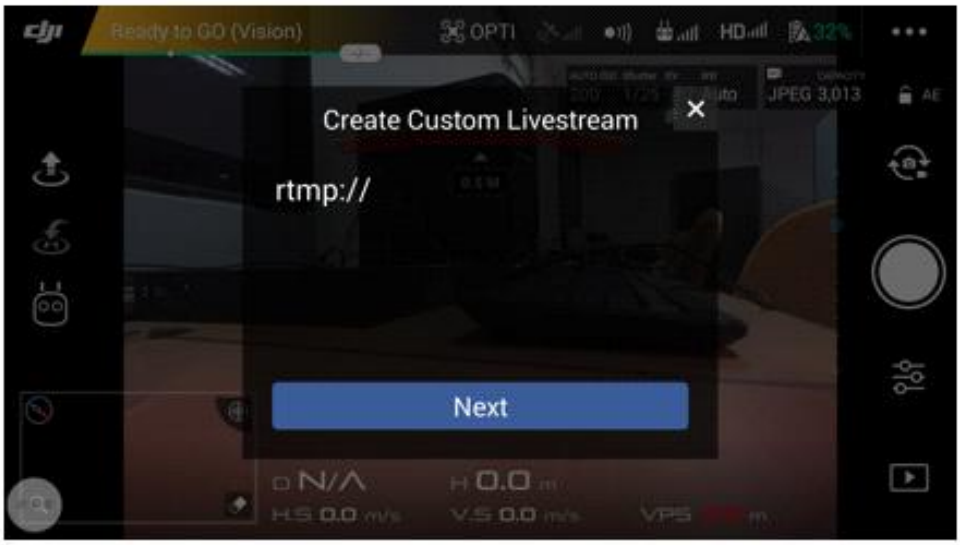


<그림 24: 드론 스트리밍 설정 방법-2>

○ Custom 선택



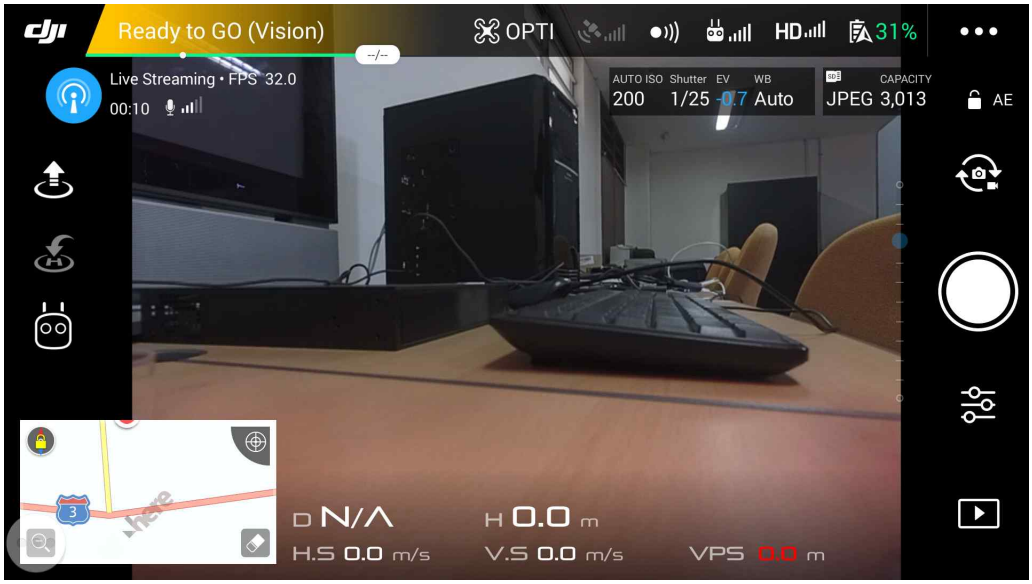
<그림 25: 드론 스트리밍 설정 방법-3>



<그림 26: 드론 스트리밍 설정 방법-4>

○ rtmp://${server_ip}:1935/encoder/${stream_name}

server_ip는 nginx가 설치되어있는 server의 ip를 입력한다. stream_name은 아무 원하는 이름을 영어로 입력하면 된다.

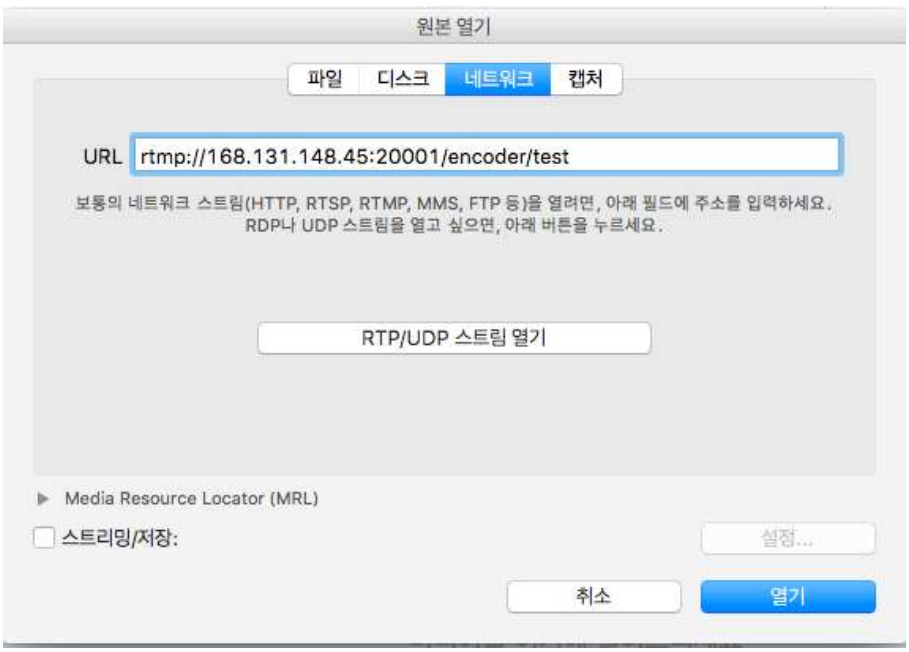


<그림 27: 드론 스트리밍 설정 방법-5>

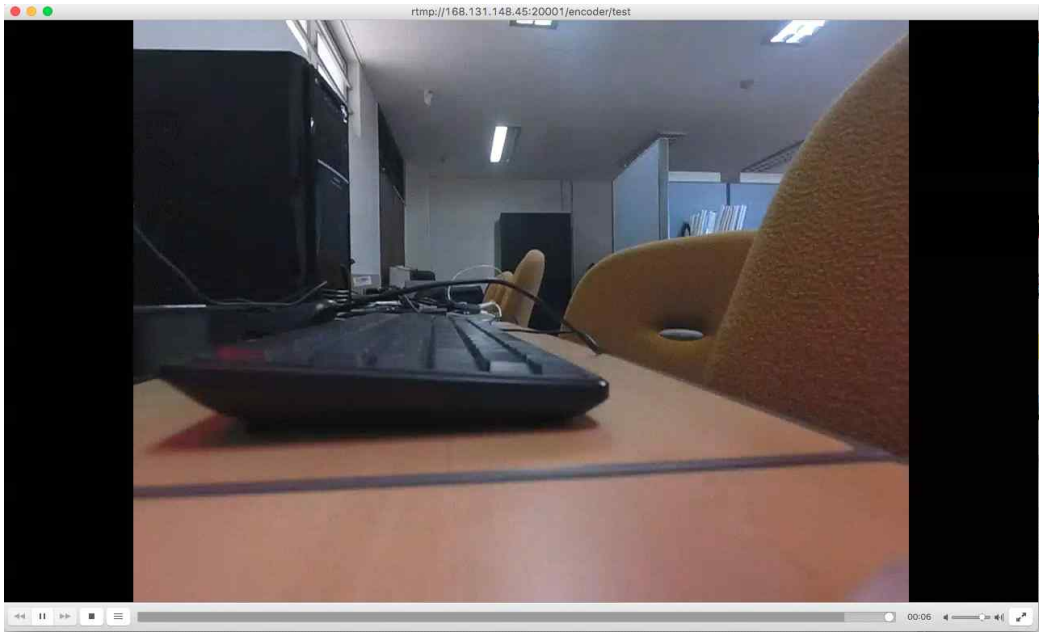
○ 왼쪽 위에 Live Streaming이 뜨고 FPS가 0이 아니다면 정상적으로 스트리밍이 진행 중이다.

4.3 검증 방법 및 결과 확인

○ rtmp를 VLC 플레이어에서 드론에서 설정한 rtmp url을 입력한 후 스트리밍데이터를 확인한다.



<그림 28: VLC를 이용한 스트리밍 테스트 url설정>

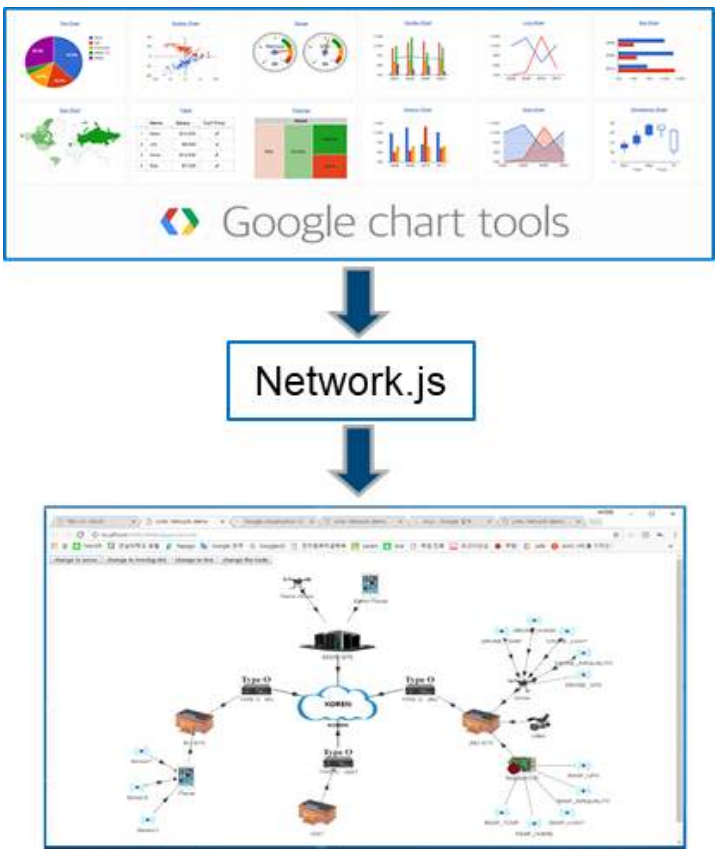


<그림 29: VLC를 이용한 스트리밍 테스트 확인>

5. Network Flow Visualization

○ 먼저 기초가 될 Network 구성도를 설계한 후 nodesTable과 LinksTables에 들어갈 속성을 정의한다.

○ 이 후, Google Chart tools을 이용하여 노드와 링크 정보를 가진 테이블을 제작한 후 Flow animation기능을 가진 network.js와 함께 Network Visualization제작한다.

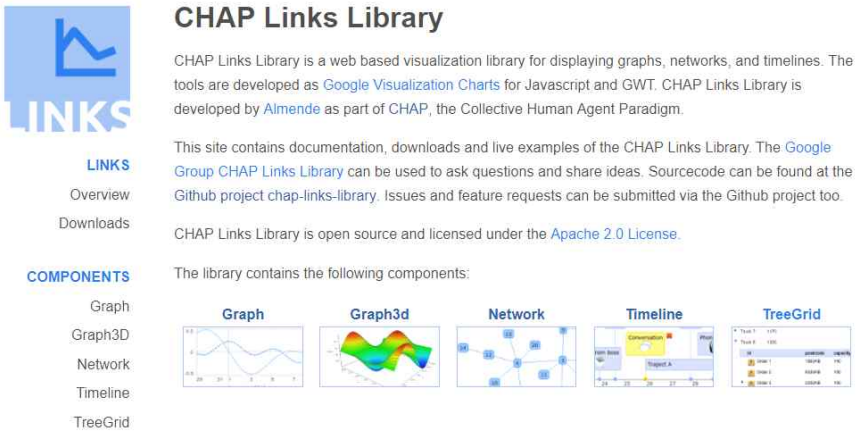


<그림 30: Network.js를 이용한 web 설계>

5.1. Network.js기반 visualization 생성

○ Flow animation 기능을 갖춘 Network Visualization을 제작하기 위해 network.js 라이브러리 다운로드하고 다운로드한 파일을 /demo-visualize/asset에 넣는다. 다운로드 url은 다음과 같다.

http://almende.github.io/chap-links-library/js/files/network-1.5.0.zip



<그림 31: CHAP Links Library 홈페이지>

5.2 Goolge Chart를 이용한 nodesTable, linksTable 생성

○ Google Visualization API Reference를 참조하여 nodesTable과 linksTable제작한다.

5.2.1. nodesTable 생성

○ nodesTable만들기

```
//make the Matrix(including column and row)
nodesTable = new google.visualization.DataTable();
```

○ nodesTable의 Column값 추가

```
//when you add the column, Column goes orderly
nodesTable.addColumn('number', 'id');
nodesTable.addColumn('string', 'text');
nodesTable.addColumn('string', 'image');
nodesTable.addColumn('string', 'style');
nodesTable.addColumn('number', 'x');
nodesTable.addColumn('number', 'y');
```

○ nodesTable의 Row값 추가

```
//when you add the row, Row goes orderly
nodesTable.addRow([1, 'KOREN', DIR + 'cloud.PNG', 'image', 500, 350]);
nodesTable.addRow([2, '', DIR + 'TYPEO-hard.PNG', 'image', 300, 300]);
nodesTable.addRow([3, '', DIR + 'TYPEO-hard.PNG', 'image', 700, 300]);
nodesTable.addRow([4, '', DIR + 'TYPEO-hard.PNG', 'image', 500, 500]);
nodesTable.addRow([5, 'KU SITE', DIR + 'university.png', 'image', 150, 400]);
nodesTable.addRow([6, 'GIST', DIR + 'university.png', 'image', 800, 200]);
```

생성된 nodesTable 예시

ID	TEXT	IMAGE
1	KOREN	cloud.PNG
2	TYPE O – KU	TYPEO-hard.PNG
3	TYPE O – GIST	TYPEO-hard.PNG
4	TYPE O – JNU	TYPEO-hard.PNG
5	KU SITE	university.png
6	GIST	university.png
7	JNU SITE	university.png
8	PHONE	Iphone.PNG
9	VIDEO	Video.PNG
10	DRONE	Drone.PNG
11	SENSOR1	Sensor.png
12	SENSOR2	Sensor.png
13	SENSOR3	Sensor.png
14	DRONE_TEMP	Sensor.png
15	DRONE_HUMID	Sensor.png
16	DRONE_LIGHT	Sensor.png
17	DEMO SITE	demo.png
18	DEMO DRONE	Drone.PNG
19	DEMO PHONE	Iphone.PNG

<그림 32: Google Chart의 nodesTable>

5.2.2. linksTable 생성

○ linksTable만들기

```
linksTable = new google.visualization.DataTable();
```

○ linksTable의 Column값 추가

```
linksTable.addColumn('number', 'from');
linksTable.addColumn('number', 'to');
linksTable.addColumn('string', 'style');
linksTable.addColumn('string', 'color');
```

○ linksTable의 Row값 추가

```
linksTable.addRow([2, 1, 'moving-arrows', undefined]);
linksTable.addRow([3, 1, 'moving-arrows', undefined]);
linksTable.addRow([4, 1, 'moving-arrows', undefined]);
linksTable.addRow([5, 2, 'moving-arrows', undefined]);
linksTable.addRow([6, 3, 'moving-arrows', undefined]);
linksTable.addRow([7, 4, 'moving-arrows', undefined]);
linksTable.addRow([8, 5, 'moving-arrows', undefined]);
linksTable.addRow([9, 7, 'moving-arrows', undefined]);
linksTable.addRow([10, 7, 'moving-arrows', undefined]);
linksTable.addRow([11, 8, 'moving-arrows', undefined]);
linksTable.addRow([12, 8, 'moving-arrows', undefined]);
```

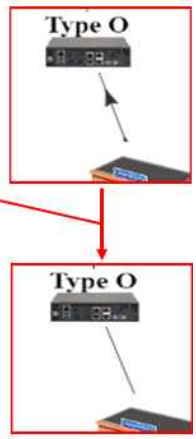
5.2.3. setCell()를 이용한 데이터 변경

○ Table안의 데이터 값 중 원하는 곳의 데이터만 변경할 수 있도록 setCell()을 사용한다. setCell()을 통해 기본적인 link모양에서 moving arrow등의 animation기능을 가진 link모양으로 변경이 가능하다.

setCell(rowIndex, columnIndex [, value [, formattedValue [, properties]]])

```
linksTable.setCell(5, 2, 'line');
network.draw(nodesTable, linksTable, options);
```

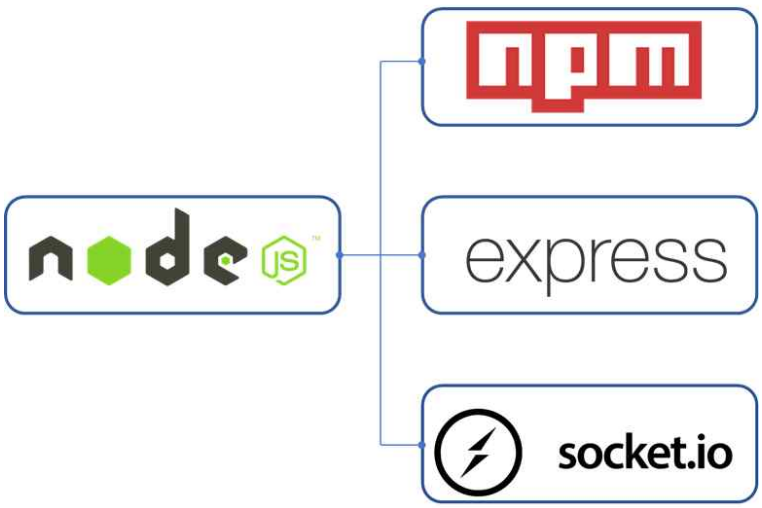
	from	to	style	color
link1	2	1	'moving-arrows'	undefined
link2	3	1	'moving-arrows'	undefined
link3	4	1	'moving-arrows'	undefined
link4	5	2	'moving-arrows'	undefined
link5	6	3	'moving-arrows'	undefined
link6	7	4	'moving-arrows'	undefined
link7	8	5	'moving-arrows'	undefined
link8	9	7	'moving-arrows'	undefined
link9	10	7	'moving-arrows'	undefined
link10	11	8	'moving-arrows'	undefined
link11	12	8	'moving-arrows'	undefined
link12	13	8	'moving-arrows'	undefined
link13	14	10	'moving-arrows'	undefined
link14	15	10	'moving-arrows'	undefined
link15	16	10	'moving-arrows'	undefined
link16	16	10	'moving-arrows'	undefined
link17	17	1	'moving-arrows'	undefined
link18	18	17	'moving-arrows'	undefined
link19	19	17	'moving-arrows'	undefined

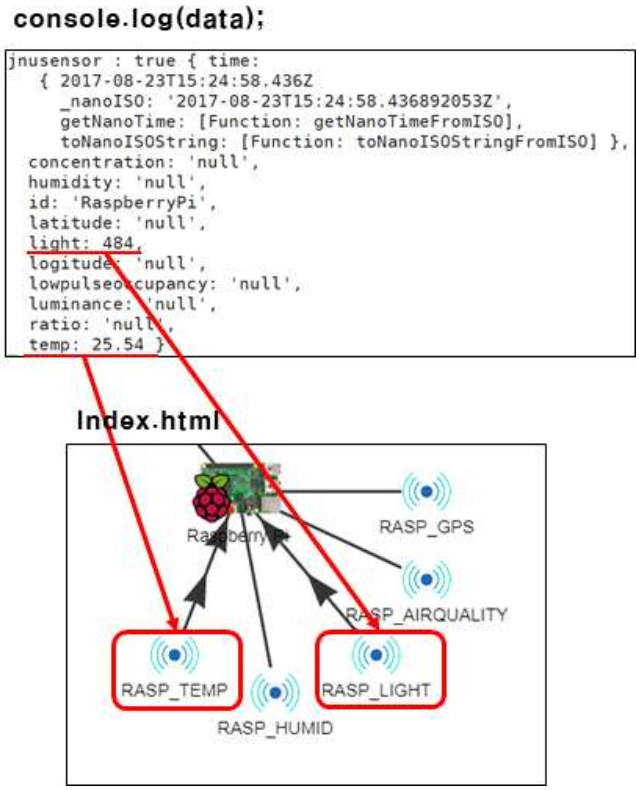


<그림 33: 데이터 변경(moving arrow)>

5.3. Socket.io를 이용한 센서 데이터 수집 여부 기능 생성 및 검증

○ node.js 설치 후 Express프레임 워크를 이용해 WAS 서버를 구축한다.





<그림 34: 센서 데이터 로그>

5.3.1. app.js의 데이터 호출

```
io.on('connection', function(socket) {

socket.on("init", function() {
DB.query(`
        select * from jnusensor
        where time > now() - 3s
        order by time desc
        limit 1
    `).then(result => {
        if(result[0] != undefined)
          raspIsOn = true;
        else
          raspIsOn2 = false;
        socket.emit('rasp', result[0]);
    }).catch(err => {
        raspIsOn = false;
        socket.emit('rasp', null)
    })
```

<그림 35: app.js의 데이터 호출>

○ connection event handler을 통해 connection이 수립되면 event handler function의 인자로 socket이 들어온다.

5.3.2 JavaScript 처리

```
socket.on("rasp2", function (data) {
console.log(data);
        if (data == null) {
} else {


if (data.temp != "null") {
linksTable.setCell(20, 3, 'moving-arrows');
}

if (data.humidity != "null") {
linksTable.setCell(21, 3, 'moving-arrows');
}

if (data.light != "null") {
linksTable.setCell(22, 3, 'moving-arrows');
}

if (data.ratio != "null" && data.concentration != "null"
    && data.lowpulseoccupancy != "null") {
linksTable.setCell(23, 3, 'moving-arrows');
}

if (data.latitude != "null" && data.logitude != "null") {
linksTable.setCell(25, 3, 'moving-arrows');
}

network.draw(nodesTable, linksTable, options);
}

});
```

<그림 36: 상황별 JavaScript 처리>

○ html 파일의 JavaScript에서 데이터가 전송되지 않을 경우를 조건으로 'arrow 모양'을 변경하여 가시적인 효과를 제공한다.